Electronic Communications of the EASST
Volume 58 (GT-VMT 2013)

Proceedings of the
12th International Workshop on Graph Transformation
and Visual Modeling Techniques
(GTVMT 2013)

Well-formed Model Co-evolution with Customizable Model Migration

Florian Mantz, Gabriele Taentzer, Yngve Lamo

13 pages

Guest Editors: Matthias Tichy, Leila Ribeiro
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Well-formed Model Co-evolution with Customizable Model Migration [*]

Florian Mantz[1], **Gabriele Taentzer**[2 1], **Yngve Lamo**[1]

[1] fma@hib.no, yla@hib.no
Institutt for data og realfag
Høgskolen i Bergen, Norway
[2] taentzer@informatik.uni-marburg.de
Fachbereich Mathematik und Informatik
Philipps-Universität Marburg, Germany

Abstract: Model-driven engineering (MDE) is a software engineering discipline which focuses on models as the primary artifact of the software development process while programs are mainly generated by means of model-to-code transformations. In particular, modeling languages tailored to specific domains promise to increase the productivity and quality of software. Nevertheless due to e.g. evolving requirements, modeling languages evolve and existing models have to be migrated. Corresponding manual model migration is tedious and error-prone, therefore tools have been developed to (partly) automate this process. We follow the idea of considering such modeling language and model co-evolutions as related graph transformations ensuring a correct and unique typing of migrated models. In this paper, we present a general and formal construction of well-formed model migration schemes that are able to co-adapt any model of a given modeling language to a performed meta-model change. We show how appropriate model migration schemes can be constructed and discuss how they may be customized.

Keywords: meta-model evolution, model migration, graph transformation

1 Introduction

Model-driven engineering [Fow10] (MDE) is a software engineering discipline which raises the abstraction level in software development by using models as primary artifacts. In particular, domain-specific modeling languages (DSMLs) are means to increase productivity and quality of software. Developers can focus on their essential tasks while repetitive and technology-dependent artifacts are automatically generated by transformations specified by experts in these areas. To keep this high level of abstraction, a modeling language has to evolve simultaneously to the evolving understanding of its target domain. However, this often causes trouble since existing models need to co-evolve with their languages (see. Figure 1). This evolution-migration problem has been tackled in practice with the result that tools have been developed that (partially) automate this tedious and error-prone process (see e.g. [HBJ09, RKPP10]). In current approaches however, parts of model migrations still have to be specified manually. In our work,

[*] This work was partially funded by NFR project 194521 (FORMGRID)

we support this step by generating default migration rule schemes for arbitrary meta-model evolution steps. These rule schemes may be customized by the user as long as the given correctness criteria are satisfied.

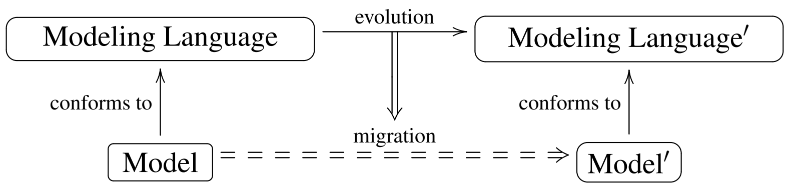


Figure 1: Model co-evolution: Modeling language evolution and model migration

Modeling languages in MDE are usually defined by meta-models. Hence, the challenge of modeling language evolution in MDE is often referred to as the problem of meta-model evolution with corresponding model migration. We tackle this model co-evolution challenge by using algebraic graph-transformations [EEPT06]. More specifically, we employ a variant of the classical double-pushout (DPO) approach using co-span rules [EHP09] offering a better synchronization of deletion and creation actions than the usual DPO approach [TML12]. For example, moving an element can be easier represented by first adding a new copy of the element before deleting its old version. In the co-span approach, the intermediate graph contains the whole context while in the span approach, it contains the preserved part only. This context can be advantageously used to formulate adequate model migrations. Our earlier work in [TML12, MTL12] presents a general framework of co-transformations by graph transformations where meta-model evolutions and model migrations are defined as inter-related graph transformations. This framework allows for migration variants. It also shows how model migrations can be derived automatically from meta-model evolutions resulting in model-specific migration rules.

In this paper, we go a step further and present model-independent migration schemes. Given an evolution rule, a default migration scheme is automatically deduced and may be customized to special needs. While the default migration scheme replays changes in the evolution rule as far as possible, potential customizations may follow special strategies to insert new model elements or to glue existing ones. Evolutions that need custom migration schemes are e.g. the insertion of a new objects of singleton classes such as registries, the insertion of new containers, new acyclic connections between model elements, new connections to existing model elements, etc. Roughly spoken, customization of migration schemes is needed whenever the insertion of new elements and gluing of existing ones need to fulfill special requirements. After the customization phase, the adapted migration scheme is applied to a specific meta-model evolution and a specific instance model. We show how a well-formed, model-specific migration rule can be deduced automatically and then applied.

2 Evolution Scenario

In the following, we consider a small example describing a co-evolution scenario of colored Petri nets [Jen03]. Figure 2 shows a simplified meta-model for colored Petri nets and an instance model, i.e. a Petri net. The Petri net variant in Figure 2 supports weighted arcs as well as colored tokens. The meta-model and the example Petri net are presented in abstract and concrete syntax. While we are working with the abstract syntax (on the right) in our theory, meta-models and models are also shown in concrete syntax (on the left) to give the reader an intuition how they are usually presented to a modeler.

To illustrate our approach, the small evolution scenario covers several steps: Petri nets shall be equipped with containers for places and transitions. The container insertion is done in two steps: First, a model container is inserted for all places and second, transitions are also put into those containers.

Third, the attribute "color" is moved from "Place" to "Token". Meta-model evolutions and model migrations are formulated by rule-based transformations, i.e. evolution and migration rules specify corresponding changes. Figure 3 shows the insertion of a new container for places as example co-transformation depicting models in abstract syntax.

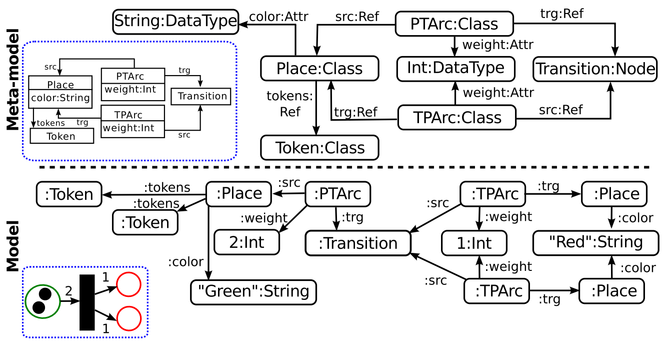


Figure 2: Petri net meta-model with an instance model

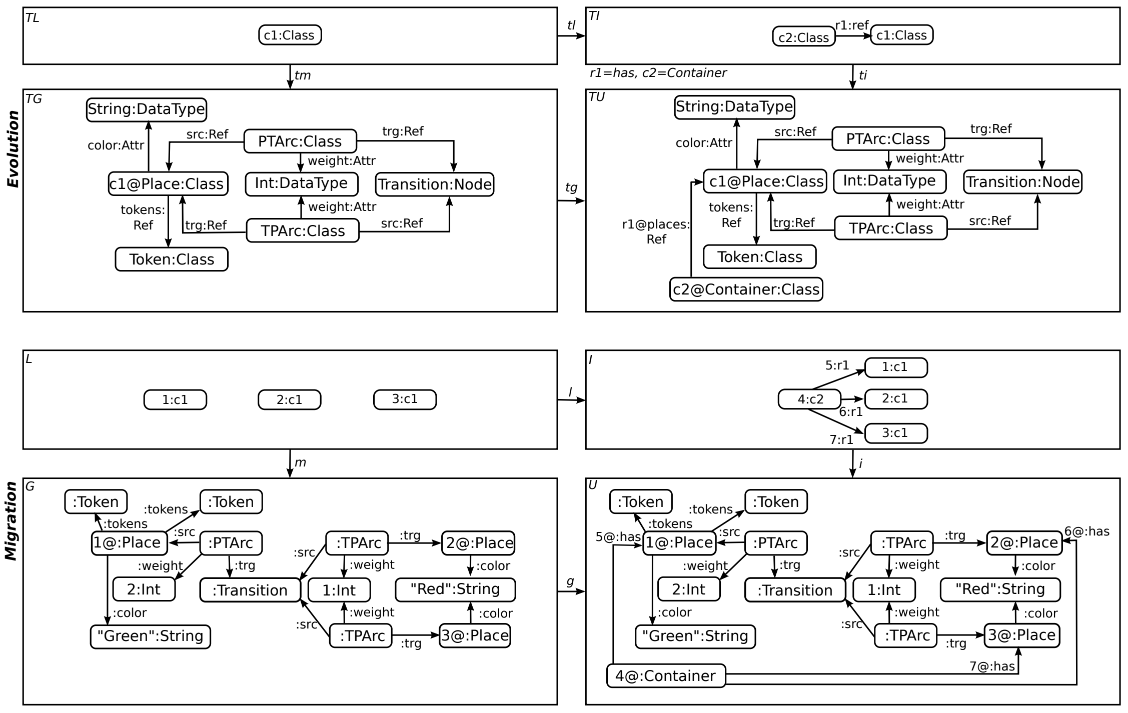


Figure 3: Co-transformation example: Insertion of a container for places

The upper layer shows the meta-model evolution where a new container for places is inserted.

The lower layer shows the migration of a specific Petri net model (introduced in Figure 2) where one container is inserted for all its places. Note that graph mappings are encoded in graph element names: Strings before "@" indicate graph mappings within one layer while strings after "@" and before '':" of meta-model elements as well as strings after ":" of instance model elements are used to indicate the typing of instance model elements (i.e. graph mappings between layers).

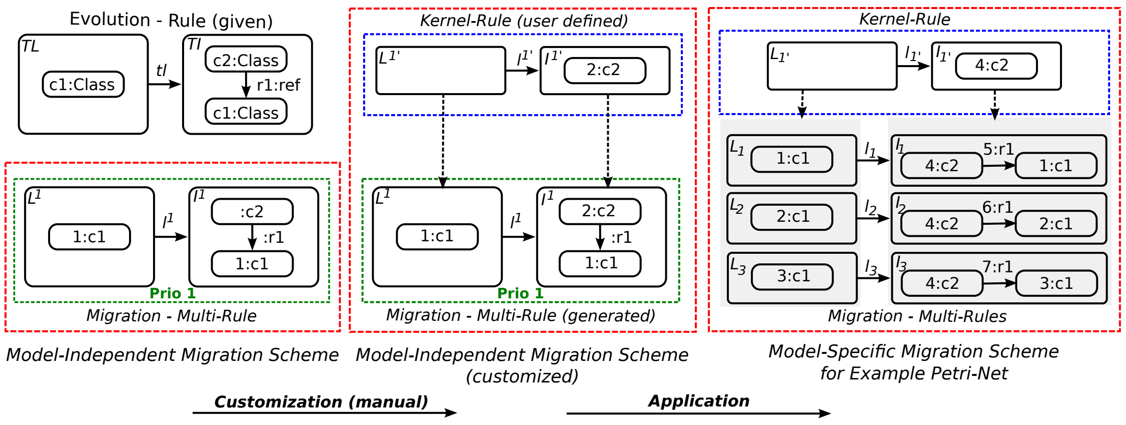


Figure 4: Migration schemes for adding a new model element

Since migration rules have to specify all model changes, they are model-specific and typically grow with increasing model sizes. Therefore, we are looking for a model-independent solution to specify model migrations. This leads us to migration schemes. An example migration scheme for the insertion of a new class is shown in Figure 4 in the lower left corner. It just consists of one basic migration rule to be applied multiple times , hence it is called a multi-rule. Migration rules are typed by evolution rules. In this example, the migration rule is isomorphic to its evolution rule. The rule specifies the basic action during migration: insertion of a new element connected to the old one. Applying this multi-rule as often as possible to all places in a Petri net would give us a new container for each match of the rule. This is not exactly what we want to have. Actually, the new element should be the same for all matches. Therefore, we customize the migration scheme to this special need and add a so-called kernel rule specifying that all new elements shall be glued to one. (See the migration scheme in the middle of Figure 4.) This customization suits well to the first evolution step where a new container (and only one) is inserted for all places. On the right, we see a migration scheme being specific for model G in Figure 3 where three places occur. Therefore, three copies of the multi-rule are in that scheme and each one is applied exactly once. Gluing all these rules at the kernel rule yields the migration rule in Figure 3.

Since the connection of elements to an existing container can be handled similarly to its insertion, it is left out here, due to space limitations. The third step is considerably different and discussed in the following. Figure 5 shows an evolution rule for moving an attribute from one class to another one. The corresponding default migration scheme contains 6 multi-rules: We start with an isomorphic copy of the evolution rule. To ensure that all elements typed by the evolution rule are matched, we need to add sub-rules for each connected component of the rule. In this case, each sub-rule is an identity rule, hence we only show their left-hand side. We prioritize the rules based on their inclusions. The one with the highest priority is an exact copy of the evolution rule. All others are identity rules and have lower priorities. Their purpose is to complete the scheme such that also parts of TL can be identified for migration and potentially deleted. For example, rule r^3 being the identity on L^3, is needed to identify all old attributes to be removed (although not moved to another class but deleted by default). Rules with higher priority are matched first. All rules in one priority class can be matched in parallel. And furthermore, a rule is matched only if its match is not already completely covered by a previous match. Applying this scheme to Petri nets, the color attribute may be moved from “Place” to “Token” and is removed from places even without connected “Token”-element to remain well-typed.

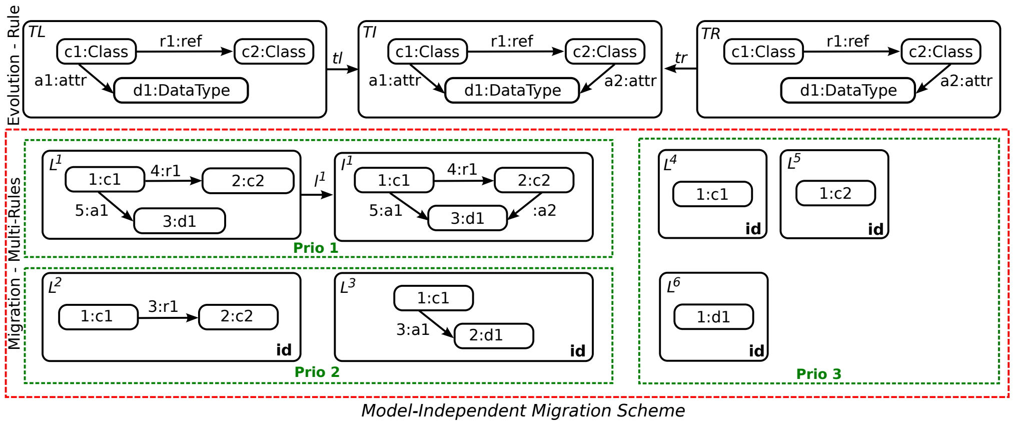


Figure 5: Migration Rule Derivation

3 Meta-model Evolution with Model Migration by Co-transformation

First, we recall the basic definitions of co-span transformations and co-transformations as defined in [EHP09, TML12, MTL12]. Co-span transformations are analog to usual DPO transformations with the exception that they use co-spans instead of spans, i.e. additions are performed before deletions. Co-transformations are the formal framework we use to formulate model co-evolution. We choose the usual category of graphs and graph morphisms as the underlying category (for details see [EEPT06]).

Definition 1 (Graph) A graph $G = (G_V, G_E, src^G, trg^G)$ consists of a set G_V of vertices (or nodes), a set G_E of edges (or arrows), and two maps $src^G, trg^G : G_E \to G_V$ assigning the source and target to each edge, respectively. $e : x \to y$ denotes an edge e with $src^G(e) = x$ and $trg^G(e) = y$.

Definition 2 (Graph morphism) A graph morphism $g : G \to H$ consists of a pair of maps $g_V : G_V \to H_V$, $g_E : G_E \to H_E$ preserving the graph structure, i.e., for each edge $e : x \to y$ in G we have an edge $g_E(e) : g_V(x) \to g_V(y)$ in H with $g_V \circ src^G = src^H \circ g_E$ and $g_V \circ trg^G = trg^H \circ g_E$.

As mentioned before, the migration procedure constructs migration schemes that rely heavily on connected components.

Definition 3 (Connected component) Given a graph H, we define an equivalence relation on vertices $\equiv\,\subseteq H_V \times H_V$ where $\equiv$ is the transitive closure of $\sim$ with $v_1 \sim v_2 \iff \exists e\colon v_1 \to v_2 \in H_E$ or $\exists e\colon v_2 \to v_1 \in H_E$. Let $[v]_\equiv \subseteq H_V$ denote an equivalence class of $\equiv$. A complete subgraph $G \sqsubseteq H$ over $[v]_\equiv$ is called a *connected component*.

Note that each vertex (and edge) belongs to exactly one connected component. Note further that injective morphisms are denoted by "$\rightarrowtail$", surjective morphisms by "$\twoheadrightarrow$", bijective morphisms by their combination, and inclusions by "$\hookrightarrow$".

Definition 4 (Co-span transformation rule) A *co-span transformation rule* $p = L \xrightarrow{l} I \xleftarrow{r} R$ consists of graphs L (left-hand-side), I (interface), and R (right-hand-side) and two jointly surjective morphisms l and r where r is injective. If r is bijective a rule p is called non-deleting and can be written as $l = L \to I$. In the case of non-deleting rules we also call I right-hand-side.

Definition 5 (Co-span transformation) Given a co-span transformation rule $p = L \xrightarrow{l} I \xleftarrow{r} R$ together with an injective morphism $m : L \rightarrowtail G$, called *match*, rule p can be applied to G if a co-span double-pushout exists as shown in the diagram on the right. $t : G \overset{p,m}{\Longrightarrow} H$ is called a *co-span transformation*.

$$\begin{tikzcd}
L \arrow[r, "l"] \arrow[d, tail, "m"'] \arrow[dr, phantom, "(PO1)"] & I \arrow[d, tail, "i"] \arrow[dr, phantom, "(PO2)"] & R \arrow[l, tail, "r"'] \arrow[d, tail, "m'"] \\
G \arrow[r, "g"'] & U & H \arrow[l, tail, "h"]
\end{tikzcd}$$

The co-span double pushout exists if the co-span gluing condition holds (see [EHP09, MTL12] for more details). Note that morphism l is allowed to be non-injective which means that graph elements may also be glued during evolutions. Co-span transformations are used to define co-transformations formalizing meta-model evolutions and their corresponding model migrations in a rule-based manner.

Definition 6 (Co-transformation rule) A co-span rule $tp = TL \xrightarrow{tl} TI \xleftarrow{tr} TR$ together with a co-span rule $p = L \xrightarrow{l} I \xleftarrow{r} R$ form a *co-transformation rule* (tp, p), if there are graph morphisms $t_L\colon L \to TL, t_I\colon I \to TI$ and $t_R\colon R \to TR$ such that both squares in the diagram on the right commute. In such a co-transformation rule (tp, p), rule tp is called an *evolution rule* while rule p is called a *migration rule* wrt. tp. We also say that migration rule p is well-typed wrt. tp.

$$\begin{tikzcd}
TL \arrow[r, "tl"] \arrow[dr, phantom, "="] & TI \arrow[dr, phantom, "="] & TR \arrow[l, tail, "tr"'] \\
L \arrow[u, "t_L"] \arrow[r, "l"'] & I \arrow[u, "t_I"] & R \arrow[u, "t_R"'] \arrow[l, tail, "r"]
\end{tikzcd}$$

Definition 7 (Match of co-transformation rule) A match (tm, m) of a co-transformation rule (tp, p) is given by two corresponding matches $tm : TL \rightarrowtail TG$ of the evolution rule tp and $m : L \rightarrowtail G$ of the migration rule p so that the matches together with the typing morphisms $t_L\colon L \to TL$ and $t_G\colon G \to TG$ construct a commuting square: $tm \circ t_L = t_G \circ m$. If $G \overset{m}{\leftarrowtail} L \xrightarrow{t_L} TL$ is a pullback then the match is called complete.

$$\begin{tikzcd}
TL \arrow[r, tail, "tm"] \arrow[dr, phantom, "="] & TG \\
L \arrow[u, "t_L"] \arrow[r, tail, "m"'] & G \arrow[u, "t_G"']
\end{tikzcd}$$

Definition 8 (Co-transformation) A co-transformation (tt, t) is defined by two graph transformations $tt : TG \overset{tp,tm}{\Longrightarrow} TH$ and $t : G \overset{p,m}{\Longrightarrow} H$ that apply a co-transformation rule (tp, p) via match (tm, m) simultaneously to type graph TG and its instance graph G, so that there are graph morphisms $t_U\colon U \to TU$ and $t_H\colon H \to TH$ such that all faces of Figure 6 commute. In such a co-transformation (tt, t), the type graph transformation $tt : TG \overset{tp,tm}{\Longrightarrow} TH$ is called an *evolution* while the instance graph transformation $t : G \overset{p,m}{\Longrightarrow} H$ is called a *migration* wrt. tt.

$$\begin{tikzcd}
TL \arrow[rr, "tl"] \arrow[dr, "tm"'] \arrow[drr, phantom, "PO1_{tt}"] & & TI \arrow[dr] \arrow[drr, phantom, "PO2_{tt}"] & & TR \arrow[ll, "tr"'] \arrow[dr] & \\
 & TG \arrow[rr] & & TU & & TH \arrow[ll] \\
L \arrow[uu, "t_L"] \arrow[rr, "l"] \arrow[dr, "m"'] \arrow[drr, phantom, "PO1_t"] & & I \arrow[uu, "t_I"] \arrow[dr] \arrow[drr, phantom, "PO2_t"] & & R \arrow[uu, "t_R"'] \arrow[ll, "r"'] \arrow[dr] & \\
 & G \arrow[uu, "t_G"] \arrow[rr] & & U \arrow[uu] & & H \arrow[uu] \arrow[ll]
\end{tikzcd}$$

Figure 6: Co-transformation

An example of a co-transformation is given in Section 2 illustrated by Figure 3. Since the example co-evolution step does not delete anything, the right cube is the identity. Therefore, the left cube is shown only.

4 Migration Schemes and Automatic Model Migration

In the following, we present a model-independent solution to specify model migrations by so-called migration schemes. The concept of migration scheme builds up on interaction schemes used to specify amalgamated graph transformation as presented in e.g. [Tae96]. For a better understanding, we recall basic definitions of kernel and multi-rules and interaction schemes, here for non-deleting rules. Thereafter, model-independent and later model-specific migration schemes are defined as special interaction schemes. Migration schemes cover the insertion and gluing of graph elements only, while their deletion is done by a pullback construction. The reason for this design decision is the observation that deletions in meta-model evolutions have to be completely reflected by their model migration while this is not true for insertions and gluings. New meta-model elements, for example, need not cause any changes in models but they may. Hence, we define a default migration scheme for a given evolution rule and allow that it may be customized under certain conditions (see Section 5 for allowed customizations).

In the following, we distinguish multi-rules to be applied as often as possible from kernel rules that are needed to glue newly inserted elements such that they are only inserted once for several multi-rules matches overlapping in the left-hand-side of the kernel rule.

Definition 9 (Kernel morphism for non-deleting rules) Given non-deleting rules $l_1 \colon L_1 \to I_1$ (called kernel rule) and $l_2 \colon L_2 \to I_2$ (called multi-rule), a kernel morphism $k \colon l_1 \to l_2$ is a pair (a,b) of morphisms $a \colon L_1 \to L_2$ and $b \colon I_1 \to I_2$ such that (1) is commuting.

$$\begin{tikzcd}
L_1 \arrow[r, "l_1"] \arrow[d, "a"'] \arrow[dr, phantom, "(1)"] & I_1 \arrow[d, "b"] \\
L_2 \arrow[r, "l_2"'] & I_2
\end{tikzcd}$$

Definition 10 (Interaction scheme of non-deleting rules) Given a set K of non-deleting kernel-rules, a set M of non-deleting multi-rules and a set KM of kernel morphisms $k \colon l_k \to l_m$ from K to M with $l_k \in K$ and $l_m \in M$. An interaction scheme of non-deleting rules is a triple (K,M,KM).

Three examples for interaction schemes of non-deleting rules are given in Figure 4. These are the model-independent migration schemes on the left and in the middle of the figure as well as the model-specific one on the right.

In the following, we construct special interaction schemes for model migrations, so-called migration schemes. Given an evolution rule $TL \to TI$, a default migration scheme is constructed by defining multi-rules for all connected subgraphs of TL and their extensions by connected parts in TI. We consider connected graphs due to the assumption that they define model parts that belong together. Including multi-rules over all subgraphs of connected components ensures that also partial patterns are recognized and adequately migrated in model-specific migration rules. In addition, loop edges in meta-models may have instances not being loops. Those cases also have to be covered by migration rules. Kernel rules are considered to be all empty by default.

Definition 11 (Default migration scheme) Given a non-deleting evolution rule $tp\colon TL \to TI$, the default migration scheme $(\overline{K}, \overline{M}, \overline{KM})_{tp}$ for rule tp is constructed by the following steps:

1. Construct a set $\overline{ML}$ of multi-rule left-hand sides by creating all possible connected subgraphs of TL. For each loop in TL, a left-hand-side is added containing a graph that has only one unfolded edge:

$$\overline{ML} := \{L \sqsubseteq TL \mid \text{ L is connected}\} \,\cup\, \overline{ML}_{loop}$$
$$\overline{ML}_{loop} := \{L_E = \{e\colon v_1 \to v_2\}, L_V = \{v_1, v_2\}\ \forall e \in TL \text{ with } src^{TL}(e) = trg^{TL}(e)\}$$

   In addition, $prio\colon \overline{ML} \to \mathbb{N}$ defines the priority function with $prio(L^i) > prio(L^j)$ if $L^i \sqsupset L^j$.

2. Construct a set of non-deleting multi-rules:
   First construct $\forall\, \overline{L} \in (\overline{ML} \setminus \overline{ML}_{loop})$ a graph I^L with:

$$I^L_V = TI_V \setminus tl(TL_V \setminus t_L(L_V))$$
$$I^L_E = \{e \in TI_E \setminus tl(TL_E \setminus t_L(L_E)) \mid src^{TI}(e) \in I^L_V \wedge trg^{TI}(e) \in I^L_V\}$$

$$\begin{tikzcd} TL \arrow[r, "tl"] & TI \\ L \arrow[u, hook, "t_L"] \arrow[r, dotted, "l"] \arrow[ur, phantom, "="] & I^L \arrow[u, hook, dotted, "t_I"'] \end{tikzcd}$$

   I^L completes $tl(L)$ by the new elements of TI.

$$\overline{M} := \{l_L\colon L \to \pi(tl(L), I^L) \mid L \in (\overline{ML} \setminus \overline{ML}_{loop}) \text{ and } l_L = tl|_{L_i}\} \,\cup\, \overline{M}_{loop}$$
$$\overline{M}_{loop} := \{id_L\colon L \to L \mid L \in \overline{ML}_{loop}\}$$

   where $\pi(U, H)$ denotes the connected component of H that contains U_V if $U \sqsubseteq H$.

3. $\forall\, l\colon L \to I \in \overline{M} : \exists in_L : L \hookrightarrow TL$ and $in_I : I \hookrightarrow TI$ being the typing morphisms.
4. Set $\overline{K} = \{\emptyset\colon \emptyset \to \emptyset\}$ contains the empty kernel rule $\emptyset$ as default.
5. $\overline{KM} = \{\emptyset_l\colon \emptyset \to l \mid \forall l \in M\}$ is the default set of kernel morphisms.

Given a default migration scheme, a meta-model evolution, and an instance model, the model-specific migration rule is constructed by matching all multi-rules as often as possible such that matches are not completely covered by other matches. Computing the intersection for each two matches, applications of multi-rules can be synchronized by common kernel rules. If available, a suitable kernel rule is selected from the given migration scheme. Otherwise, it is constructed by epi-mono-factorization. This means that the intersection graph is taken as left-hand side and its potential gluing in its multi-rules is reflected in the right-hand side of its kernel rule. The result of this construction is a model-specific migration scheme containing multi-rules as often as they can be matched. For an example, consider the scheme on the right of Figure 4.

Definition 12 (Model-specific migration scheme for graphs) Let $(\overline{K},\overline{M},\overline{KM})_{tp}$ be a default migration scheme over evolution rule $tp = (TL \xrightarrow{tl} TI \overset{tr}{\leftarrowtail} TR)$ and let $tt : TG \overset{tp,tm}{\Longrightarrow} H$ be an evolution step. Furthermore, let G be a graph typed by TG, i.e.$t_G\colon G \to TG$:

1. Match all multi-rules $l^u\colon L^u \to I^u \in \overline{M}$ to G and construct match set $MM =: \{m_i : L_i \rightarrowtail G \mid 1 \le i \le n\}$ such that:

   $$\forall L_i \text{ with } 1 \le i \le n \implies \exists l^u\colon L^u \to I^u \in \overline{M},\ L_i = L^u$$
   $$\text{and } \nexists\, m_j \in MM \text{ with } m_i(L_i) \sqsubseteq m_j(L_j)$$

$$\begin{tikzcd}
TG & TL \arrow[l, tail, "tm"'] & \\
G \arrow[u] \arrow[ur, phantom, "="] & L_i \arrow[l, tail, "m_i"'] \arrow[u, tail, "t_{L_i}"'] & \\
& & L_j \arrow[ul, tail, "\nexists m_j"] \arrow[ull, tail, bend left] \arrow[uul, tail, bend right, "t_{L_j}"']
\end{tikzcd}$$

2. Let $L_i \overset{a_{ij}}{\leftarrowtail} L_{ij} \overset{a_{ji}}{\rightarrowtail} L_j$ be the pullback of $L_i \overset{m_i}{\rightarrowtail} G \overset{m_j}{\leftarrowtail} L_j$, $\forall 1 \le i < j \le n$. $\forall L_{ij}$ with $1 \le i < j \le n$:

   (a) If $\exists\, l^s\colon L^s \to I^s \in \overline{K}$ with $L_{ij} = L^s$ and
   $\exists (a\colon L^s \to L^u, b\colon I^s \to I^u) \in \overline{KM}$ and $a_{ij} = a,\ L_i = L^u$.
   $\implies k_{ij} = (a_{ij}, b_{ij}) \in KM$ with $b_{ij} = b$ and $l_{ij}\colon L_{ij} \to I_{ij}$ with $I_{ij} = I^s$ be in K. (Analogously for k_{ji}). $(k_{ij}, k_{ji}) \in KM$

$$\begin{tikzcd}
L^s \arrow[r, "l^s"] \arrow[d, tail] \arrow[ddd, tail, bend right=40, "a^{uv} = a_{ij}"'] \arrow[dr, phantom, "="] & I^s \arrow[d, tail] \arrow[ddd, tail, bend left=40, "b_{ij} = b^{uv}"] \\
L_{ij} \arrow[r, dotted, "l_{ij}"] \arrow[d, tail, "a_{ij}"] \arrow[dr, phantom, "="] & I_{ij} \arrow[d, tail, dotted, "b_{ij}"'] \\
L_i \arrow[r, "l_i"] \arrow[dr, phantom, "="] & I_i \\
L^u \arrow[r, "l^u"] \arrow[u, tail, two heads] & I^u \arrow[u, tail, two heads]
\end{tikzcd}$$

   (b) Otherwise generate a kernel rule:
   Construct $(I_{ij},\ l_{ij}\colon L_{ij} \to I_{ij},\ b_{ij}\colon I_{ij} \to I_i)$ by epi-mono-factorization of $l_i \circ a_{ij}$. In this case, $l_{ij}\colon L_{ij} \to I_{ij} \in K$ and $k_{ij} = (a_{ij}, b_{ij})$. (Analogously for $k_{ji} \in KM$). Then, $(k_{ij}, k_{ji}) \in KM$

$$\begin{tikzcd}
L_{ij} \arrow[r, dotted, "l_{ij}"] \arrow[d, tail, "a_{ij}"'] \arrow[dr, "l_i \circ a_{ij}"] & I_{ij} \arrow[d, tail, dotted, "b_{ij}"] \\
L_i \arrow[r, "l_i"] & I_i
\end{tikzcd}$$

   This yields the model-specific migration scheme $(K,M,KM)_{tt,G}$.

Remark 1 *Note that the epi-mono-factorization of $l_i \circ a_{ij}$ yields the same $l_{ij}\colon L_{ij} \to I_{ij}$ as that of $l_j \circ a_{ji}$ does.*

Remark 2 *It is obvious that both kinds of migration schemes are interaction schemes of non-deleting rules.*

Having a model-specific migration scheme at hand, it can be glued by so-called star gluings yielding its model-specific migration rule. The basic idea is to glue all multi-rules along their kernel rules. Such a gluing diagram can be a kind of star in general. Therefore, we re-call the star gluing of graphs. It is shown in [Tae96] that the star gluing is a special form of co-limit construction.

Definition 13 (Star gluing)

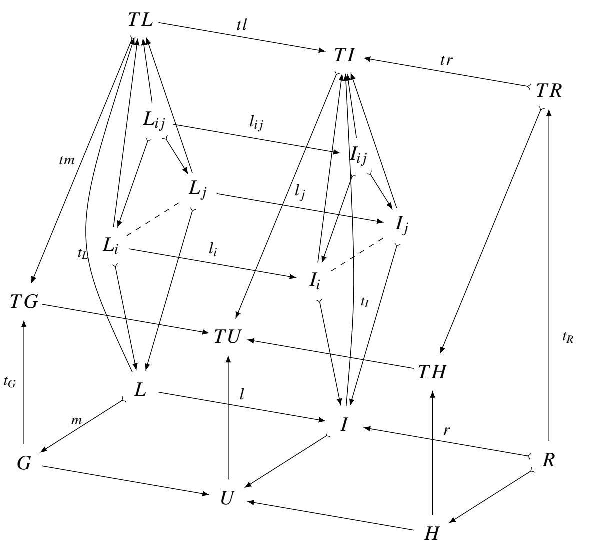
1. Given a set of graph morphism spans $S = (G_i \overset{g_{ij}}{\leftarrow} G_{ij} \overset{g_{ji}}{\rightarrow} G_j)_{1\le i<j\le n}$, a gluing relation $\bar{I}_S$ is the equivalence relation generated by the relation $I_S = \{(g_{ij}(x), g_{ji}(x)) | x \in G_{ij}, 1 \le i < j \le n\}$.
2. Given a set of graph morphism spans $S = (G_i \overset{g_{ij}}{\leftarrow} G_{ij} \overset{g_{ji}}{\rightarrow} G_j)_{1\le i<j\le n}$ with its gluing relation $\bar{I}_S$, the star gluing graph G is defined by $G = (\uplus_{1\le i\le n} G_i)_{/\bar{I}_{MS}}$, the quotient set of the disjoint union of all G_i. The maps $g_i \colon G_i \to G$ for all $1 \le i \le n$ send each element of G_i to its equivalence class in G. The pair $(G, \{g_i | 1 \le i \le n\})$ is called the *star gluing of S*.

$$\begin{tikzcd}
 & G_{ij} \arrow[dl] \arrow[dr] & \\
G_i \arrow[dr, "g_i"'] & = & G_j \arrow[dl, "g_j"] \\
 & G &
\end{tikzcd}$$

Definition 14 (Model-specific migration rule) Given a model-specific migration scheme $(K,M,KM)_{tt,G}$ with match set $MM =: \{m_i : L_i \rightarrowtail G \mid 1 \le i \le n\}$ over evolution $tt : TG \overset{tp,tm}{\Longrightarrow} TH$ as defined in Definition 12. Let G be a graph typed by TG, i.e $t_G \colon G \to TG$:

1. Take all a_{ij} and a_{ji} as well as b_{ij} and b_{ji} for $1 \le i < j \le n$ and construct the star gluings $(L, \{a_i | 1 \le i \le n\})$ and $(I, \{b_i | 1 \le i \le n\})$ of morphism stars $S_L = (L_i \overset{a_{ij}}{\leftarrowtail} L_{ij} \overset{a_{ji}}{\rightarrowtail} L_j)_{1\le i<j\le n}$ and $S_I = (I_i \overset{b_{ij}}{\leftarrowtail} I_{ij} \overset{b_{ji}}{\rightarrowtail} I_j)_{1\le i<j\le n}$ with induced morphism $l \colon L \to I$. Match m of model-specific migration rule l is the induced morphism $m : L \rightarrowtail G$ over $MM =: \{m_i : L_i \rightarrowtail G \mid 1 \le i \le n\}$.
2. Finally, construct the right-hand side R of the model-specific migration rule as pullback $I \overset{r}{\leftarrowtail} R \overset{t_R}{\rightarrow} TR$ of $I \overset{t_I}{\rightarrow} TI \overset{tr}{\leftarrowtail} TR$ (see Figure 6). r is the induced morphism of pullback $I \overset{r}{\leftarrowtail} R \overset{t_R}{\rightarrow} TR$.

The figure on the right shows the double cube of Figure 6 using a migration scheme to construct the non-deleting part of model-specific migration rules. Morphism $r : I \rightarrowtail R$ of the migration rule is constructed by a pullback deleting all instances that cannot be typed over TR. This new construction of migration rules fits to our earlier work in [TML12, MTL12] as shown in the next propositions. Hence, a model migration constructed in such a way, is applicable and migrates a model "correctly" (i.e. in a well-typed manner) and in a unique way.



Proposition 1 *A model-specific migration rule is typed over its evolution rule.*

Proof sketch: Since all multi-rules of migration schemes are well-typed (see Definition 11) and L and I are star gluing graphs, their typing morphisms t_L and t_I are induced morphisms. In addition, we have $t_I \circ l = tl \circ t_L$, due to uniqueness of induced morphisms. The existence of t_R and $t_I \circ r = tr \circ t_R$ holds due to the pullback construction.

Proposition 2 *The construction of model-specific migration rules results in complete matches only, i.e.* $TL \xleftarrow{t_L} L \overset{m}{\rightarrowtail} G$ *is the pullback of* $TL \overset{tm}{\rightarrowtail} TG \xleftarrow{t_G} G$ *(see Figure 6).*

Proof sketch: All multi-rule matches in MM induce a unique morphism $m : L \rightarrowtail G$. Since the default migration scheme contains a multi-rule for each connected subgraph of TL, all elements of TL that have occurrences in G are matched by multi-rules. Hence, the star gluing graph of the left-hand sides of a model-specific migration scheme covers at least the pullback graph. Since the overlapping of each two multi-rule matches has to be a pullback, the star gluing graph cannot be larger than the pullback graph.

5 Migration Scheme Customizations

Although constructed along reasonable design decisions, default migration schemes do not always define desired model migrations. As mentioned in Section 2, the insertion of new connected elements have to be limited in some cases, e.g. in case that a singelton class are specified. Sometimes they should be glued as in this example, sometimes they should be inserted in certain contexts only. Similarly, the gluing of meta-model elements can lead to retyping or gluing of model elements during migration but do not have to. Hence variations in migration strategies are possible and it is essential that migration schemes are customizable. However, we want to make sure that such customized migration schemes still satisfies Proposition 1 and Proposition 2. This means that the set of multi-rules is complete so that it can amalgamate every possible L and that the resulting migration rules are always well-typed over their evolution rules. Given a default migration scheme, the following customizations are possible:

1. A multi-rule may be deleted if its LHS can be amalgamated by LHSs of other multi-rules.
2. A non-deleting multi-rule may be added if its L and its I are typed over TL and TI, respectively as long as the multi-rule is not conflicting with any other rule. A conflict between two rules occurs if two elements of L are preserved by one rule and merged by the other.
3. Graph I of a multi-rule may be extended if its extension is still typed over TI.
4. A kernel rule $k : L^k \to I^k$ with kernel morphism to at least one multi-rule may be added. This means that $I^k \sqsubseteq I^m$ if $L^k \sqsubseteq L^m$ with $m : L^m \to I^m$ being the corresponding multi-rule.

An example for a customized migration scheme is given in the middle of Figure 4. Here, a new kernel rule with kernel morphism to the existing multi-rule are added.

6 Related Work

Co-evolution of structures has been considered in several areas of computer science such as for database schemes, grammars, and meta-models [Li99, Läm01, SK04]. Especially database schema evolution has been a subject of research in the last decades. Recently, research activities have started to consider meta-model evolution and to investigate the transfer of schema evolution concepts to meta-model evolution (see e.g. [HBJ09]). Herrmannsdoerfer et al. provides an overview of approaches in [HVW10] that considers the coupled evolution of meta-models and models. Inspired by the achievements for databases, current approaches for meta-model evolution use libraries and catalogs for co-evolution operations. However, meta-model evolution researchers have realized that a fixed set of co-evolution operators is not sufficient and allow therefore to extend or adapt the default migration transformations.

There are several tools around that consider the co-evolution of Eclipse Modeling Framework (EMF) models. COPE (now:Edapt) [HBJ09] provides a rich set of co-evolution operators that implement evolution and migration definitions. If the required migration is not available, a modeling language designer has to add the migration strategy as a Groovy/Java program. Epsilon Flock [RKPP10] is another tool that is able to migrate EMF models. In contrast to COPE, Flock does not provide migration operations. Instead, it uses a “conservative copy” strategy that automatically copies as much as possible to a migrated version of the model. However, migration scripts have to be written if model elements shall be moved and not forgotten, while pull up and push down attribute are correctly migrated by the conservative copy strategy. EMF Migrate [EMF] is another migration tool for EMF. It supports not only on model migration but also the migration of other meta-model-dependent artifacts such as ATL transformations. For model migration, EMF Migrate provides migration libraries that can be customized and extended. Our work differs from this related work in the sense that we consider correctness criteria on a formal level to be applied to show correctness of co-transformations in general as well as for user adaptations. Furthermore, we develop a strategy for deriving model migration schemes that are not only working for pre-defined migration operations but allow user-defined meta-model evolution rules with deduced migration schemes.

The closest work to ours is the one by König et al. [KLS11]. They also consider correctness criteria for model co-evolution based on a categorical framework. However, they do not work with algebraic graph transformation and do not provide customizable rule schemes for model migration.

7 Conclusion and Future Work

The evolution of meta-models along well-defined transformation steps does not always yield meaningful model migrations automatically. While default migration schemes can be automatically deduced from evolution rules, it is up to language designers to customize them to their specific needs. In this paper, we clarify how migration schemes can be defined and how they may be customized to still yield well-defined co-transformations of models and meta-models. Model-specific migration rules are shown to be automatically constructible from a migration scheme, a meta-model evolution, and an instance model. In the future, we want to extend this work on model migration schemes to more powerful forms of graphs and rules taking especially node type inheritance, application conditions and constraints into account. Furthermore, we want to consider further co-evolution examples to evaluate available forms of supported model migration schemes and their customization facilities. Moreover, we started to a prototype implementation of our co-evolution approach being very close to its formal foundation and hence, supporting well-formed model migrations.

Bibliography

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, March 2006.

[EHP09] H. Ehrig, F. Hermann, U. Prange. Cospan DPO Approach: An Alternative for DPO Graph Transformation. *EATCS Bulletin* 98:139–149, 2009.

[EMF] EMF Migrate. Project Web Site. http://www.emfmigrate.org.

[Fow10] M. Fowler. *Domain-Specific Languages*. Addison-Wesley Professional, 2010.

[HBJ09] M. Herrmannsdoerfer, S. Benz, E. Jürgens. COPE - Automating Coupled Evolution of Metamodels and Models. In Drossopoulou (ed.), *ECOOP 2009*. LNCS 5653, pp. 52–76. Springer, 2009.

[HVW10] M. Herrmannsdoerfer, S. Vermolen, G. Wachsmuth. An Extensive Catalog of Operators for the Coupled Evolution of Metamodels and Models. In Malloy et al. (eds.), *SLE 2010*. LNCS 6563, pp. 163–182. Springer, 2010.

[Jen03] K. Jensen. *Coloured Petri Nets*. Springer, 2003.

[KLS11] H. König, M. Löwe, C. Schulz. Model Transformation and Induced Instance Migration: A Universal Framework. In Silva Simão and Morgan (eds.), *SBMF 2011*. LNCS 7021, pp. 1–15. Springer, 2011.

[Läm01] R. Lämmel. Grammar Adaptation. In Oliveira and Zave (eds.), *FME 2001*. LNCS 2021, pp. 550–570. Springer, 2001.

[Li99] X. Li. A Survey of Schema Evolution in Object-Oriented Databases. In *TOOLS 1999*. Pp. 362–371. IEEE Computer Society, 1999.

[MTL12] F. Mantz, G. Taentzer, Y. Lamo. Co-Transformation of Type and Instance Graphs Supporting Merging of Types with Retyping: Long Version. Technical report, Department of Mathematics and Computer Science, University of Marburg, Germany, September 2012. www.uni-marburg.de/fb12/forschung/berichte/berichteinformtk.

[RKPP10] L. Rose, D. Kolovos, R. F. Paige, F. A. C. Polack. Model Migration with Epsilon Flock. In Tratt and Gogolla (eds.), *ICMT 2010*. LNCS 6142, pp. 184–198. Springer, 2010.

[SK04] J. Sprinkle, G. Karsai. A Domain-Specific Visual Language for Domain Model Evolution. *Journal of Visual Languages and Computing* 15(3–4):291–307, 2004.

[Tae96] G. Taentzer. *Parallel and Distributed Graph Transformation: Formal Description and Application to Communication-Based Systems*. PhD thesis, Technical University of Berlin, 1996.

[TML12] G. Taentzer, F. Mantz, Y. Lamo. Co-Transformation of Graphs and Type Graphs With Application to Model Co-Evolution. In Ehrig et al. (eds.), *ICGT 2012*. LNCS 7562, pp. 326–340. Springer, 2012.